# Serialization (C#)

# <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>

Serialization is the process of **converting an object** into a **stream of bytes** to **store the object or transmit it to memory, a database, or a file**. Its **main purpose is to save** the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

## **How serialization works**

This illustration shows the overall process of serialization.



The object is serialized to a stream, which carries not just the data, but information about the object's type, such as its version, culture, and assembly name. From that stream, it can be stored in a database, a file, or memory.

### **Uses for serialization**

Serialization allows the developer to save the state of an object and recreate it as needed, providing storage of objects as well as data exchange. Through serialization, a developer can perform actions like sending the object to a remote application by means of a Web Service, passing an object from one domain to another, passing an object through a firewall as an XML string, or maintaining security or user-specific information across applications.

### **Making an object serializable**

To serialize an object, you need the object to be serialized, a stream to contain the serialized object, and a [Formatter](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.formatter). [System.Runtime.Serialization](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization) contains the classes necessary for serializing and deserializing objects.

Apply the [SerializableAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.serializableattribute) attribute to a type to indicate that instances of this type can be serialized. An exception is thrown if you attempt to serialize but the type doesn't have the [SerializableAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.serializableattribute)attribute.

If you don't want a field within your class to be serializable, apply the [NonSerializedAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.nonserializedattribute) attribute. If a field of a serializable type contains a pointer, a handle, or some other data structure that is specific to a particular environment, and the field cannot be meaningfully reconstituted in a different environment, then you may want to make it nonserializable.

If a serialized class contains references to objects of other classes that are marked [SerializableAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.serializableattribute), those objects will also be serialized.

## **Binary and XML serialization**

You can use binary or XML serialization. In binary serialization, all members, even members that are read-only, are serialized, and performance is enhanced. XML serialization provides more readable code, and greater flexibility of object sharing and usage for interoperability purposes.

### **Binary serialization**

Binary serialization uses binary encoding to produce compact serialization for uses such as storage or socket-based network streams.

### **XML serialization**

XML serialization serializes the public fields and properties of an object, or the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document. XML serialization results in strongly typed classes with public properties and fields that are converted to XML. [System.Xml.Serialization](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization) contains the classes necessary for serializing and deserializing XML.

You apply attributes to classes and class members to control the way the [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer) serializes or deserializes an instance of the class.

### **Custom serialization**

In custom serialization, you can specify exactly which objects will be serialized and how it will be done. The class must be marked [SerializableAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.serializableattribute) and implement the [ISerializable](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.iserializable) interface.

If you want your object to be deserialized in a custom manner as well, you must use a custom constructor.

## Designer serialization

Designer serialization is a special form of serialization that involves the kind of object persistence associated with development tools. Designer serialization is the process of converting an object graph into a source file that can later be used to recover the object graph. A source file can contain code, markup, or even SQL table information.

## Related Topics and Examples

[Walkthrough: Persisting an Object in Visual Studio (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/walkthrough-persisting-an-object-in-visual-studio)  
Demonstrates how serialization can be used to persist an object's data between instances, allowing you to store values and retrieve them the next time the object is instantiated.

[How to: Read Object Data from an XML File (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/how-to-read-object-data-from-an-xml-file)  
Shows how to read object data that was previously written to an XML file using the [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer) class.

[How to: Write Object Data to an XML File (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/how-to-write-object-data-to-an-xml-file)  
Shows how to write the object from a class to an XML file using the [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer) class.